AD_____

Award Number: DAMD17-01-1-0825

TITLE: Zero Trust Intrusion Containment for Telemedicine

PRINCIPAL INVESTIGATOR: Arun K. Sood, Ph.D.
Yih Huang
Robert Simon
Elizabeth White
Kevin Cleary

CONTRACTING ORGANIZATION: George Mason University
Fairfax, Virginia 22030

REPORT DATE: December 2002

TYPE OF REPORT: Final

PREPARED FOR: U.S. Army Medical Research and Materiel Command
Fort Detrick, Maryland 21702-5012

REPORT DOCUMENTATION PAGE

Form Approved
OMB No. 074-0188

1. AGENCY USE ONLY (Leave blank)	2. REPORT DATE	3. REPORT TYPE AND DATES COVERED
	December 2002	Final (1 Sep 01 - 30 Nov 02)

4. TITLE AND SUBTITLE
Zero Trust Intrusion Containment for Telemedicine

5. FUNDING NUMBERS
DAMD17-01-1-0825

6. AUTHOR(S) :
Arun K. Sood, Ph.D., Yih Huang, Robert Simon, Elizabeth White, Kevin Cleary

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)
George Mason University
Fairfax, Virginia 22030

Email: asood@gmu.edu

8. PERFORMING ORGANIZATION REPORT NUMBER

9. SPONSORING / MONITORING AGENCY NAME(S) AND ADDRESS(ES)
U.S. Army Medical Research and Materiel Command
Fort Detrick, Maryland 21702-5012

10. SPONSORING / MONITORING AGENCY REPORT NUMBER

11. SUPPLEMENTARY NOTES
Original contains color plates: All DTIC reproductions will be in black and white.

12a. DISTRIBUTION / AVAILABILITY STATEMENT
Approved for Public Release; Distribution Unlimited

12b. DISTRIBUTION CODE

13. ABSTRACT ***(Maximum 200 Words)***

none provided

14. SUBJECT TERMS:
intrusion management systems (IMS), self-cleansing intrusion tolerance (SCIT)

15. NUMBER OF PAGES
26

16. PRICE CODE

17. SECURITY CLASSIFICATION OF REPORT	18. SECURITY CLASSIFICATION OF THIS PAGE	19. SECURITY CLASSIFICATION OF ABSTRACT	20. LIMITATION OF ABSTRACT
Unclassified	Unclassified	Unclassified	Unlimited

Table of Contents

Zero-trust Intrusion Containment for Telemedicine

EXECUTIVE SUMMARY

Intrusion Management Systems (IMS) serve to protect complex computer systems from unauthorized intrusions. Our objective is the design and analysis of "zero-trust" Intrusion Tolerant Systems. These are systems built under the extreme assumption that all intrusion detection techniques will eventually fail.

Our approach, Self-Cleansing Intrusion Tolerance (SCIT), centers around two key concepts: zero trust and system self-cleansing. Our zero trust principle assumes that a successful intrusion may have taken place on any part of the system that is currently running. System self-cleansing involves both hardware and software elements that periodically restore themselves from a trusted source. Our goal is to provide increased resistance to intrusions with minimal disruption of the services provided by the overall system.

This final report summarizes our research. This research project introduces a new approach to information security that we call Self-Cleansing Intrusion Tolerance (SCIT). System self-cleansing involves both hardware and software components that periodically restore themselves from a trusted source. We began this work by focusing on one key component of such a system – a SCIT firewall. Thereafter we built a SCIT web server. To confirm applicability of SCIT to the telemedicine arena, we performed perceptual testing and evaluating the effects of SCIT on packet loss and perceived performance.

Our research has accomplished the following:

1. Explored key issues in the implementation of SCIT such as the frequency of self-cleansing, the use of a SCIT certificate for authentication, and trust and dynamic reconfiguration.
2. Developed a demonstration of SCIT firewall and SCIT web server.
3. Investigated other SCIT components including a network file system (NFS) server and a certification server and client to determine suitability for SCITization.

1. INTRODUCTION

Intrusion Management Systems (IMS) serve to protect complex computer systems from unauthorized intrusions. The traditional IMS approaches rely on intrusion prevention and detection, followed by implementation of intrusion resistance procedures. A key assumption of a traditional IMS is that it is possible to detect all intrusions.

We believe that the sophistication and rapid evolution of information warfare requires the more pessimistic assumption that undetected intrusions will occur and must be guarded against as well. Our approach, Self-Cleansing Intrusion Tolerance (SCIT), centers around two key concepts: zero trust and system self-cleansing. Our zero trust principle assumes that a successful intrusion may have taken place on any part of the system that is currently running. System self-cleansing involves both hardware and software elements that periodically restore themselves from a trusted source. Our goal is to provide increased resistance to intrusions with minimal disruption of the services provided by the overall system. Our approach both complements and strengthens existing IMS approaches, by adding another layer of defense.

In the current research we have adopted a two pronged approach:

1. Explore the technical issues that will ensure the applicability of SCIT is a broad class of scenarios.
2. Build a SCIT demonstrator for a key part of the security system. Our current focus has been on a SCIT firewall.

In this progress report we provide a summary of the status of our research and present our future plan of work. The rest of the report is divided into 5 parts. Section 2 provides the objective of this project and Section 3 the contextual framework for the research. Our research effort is proceeding in two tracks, and the results and development efforts are presented in Sections 4 and 5. Section 4 treats the various factors that impact on the intrusion containment strategy that has been adopted in this research. We highlight the importance of issues such as the frequency at which the self cleansing operation should be conducted, the use of a SCIT certificate for authentication and assurance that the necessary self-cleansing operation was done recently, formulation of trust and dynamic reconfiguration to support SCIT. Section 5 focuses on the prototype system that we have built. We discuss the objective of the planned demonstration, the architecture of the prototype including the firewall and secure tunneling features, and the test approach. The demo is operational and the client station is being used to surf the web on a regular basis, and an example is provided. Future demo plans are provided and we also discuss the future development paths for this system, and in this way we show the link between the prototype development and the research mentioned in Section 4. The final section (Section 6) provides a summary of the future plans for this research effort.

2. OBJECTIVE

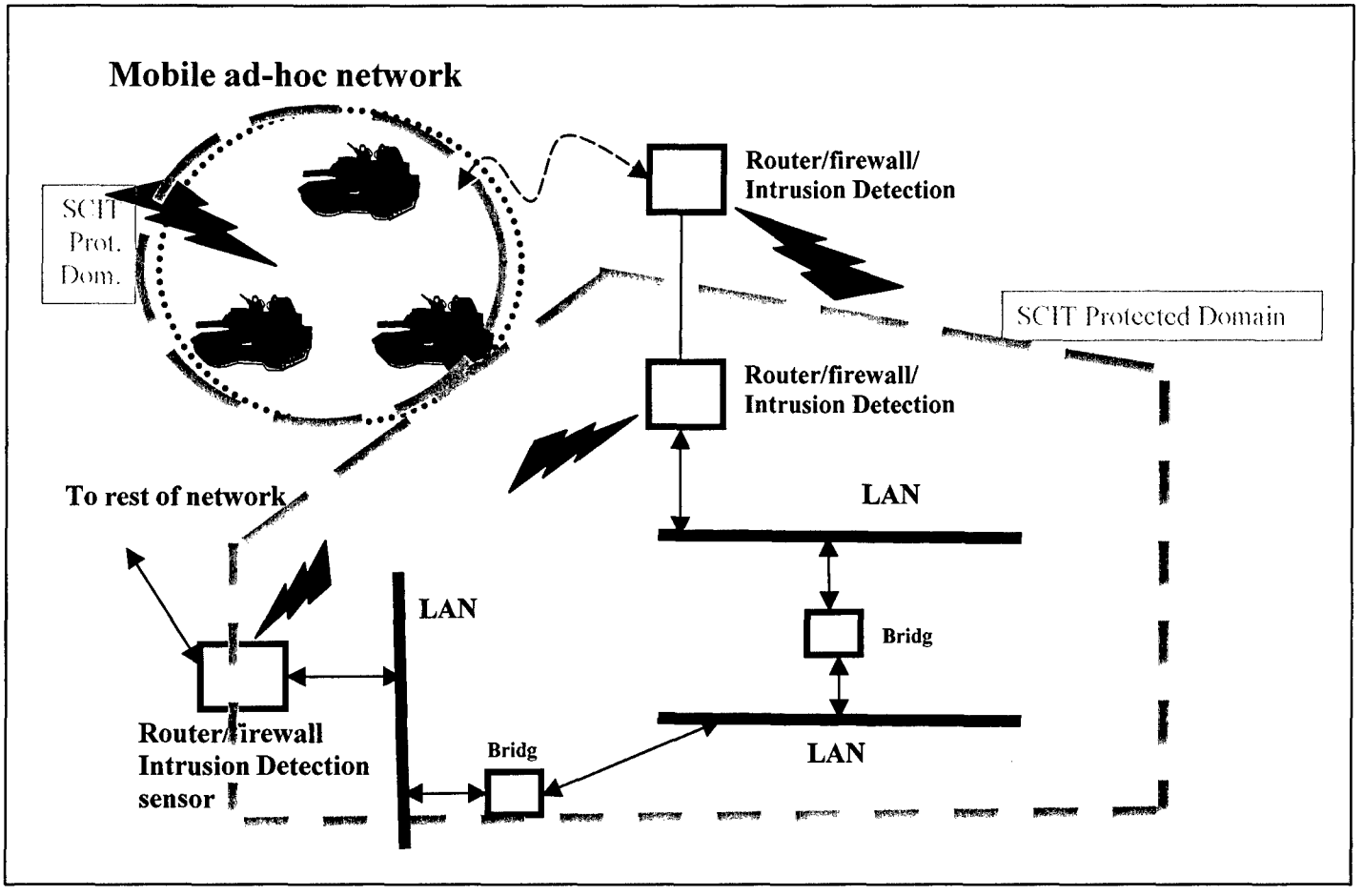
Our objective is the design and analysis of "zero-trust" Intrusion Tolerant Systems. These are systems built under the extreme assumption that all intrusion detection techniques will eventually fail. For this project, our objective is to explore a completely new approach for Intrusion Tolerance and information assurance that guarantees correct functioning of critical system features even under the certainty of successful cyber-based attack. Our approach assumes that all system components will eventually be corrupted and therefore need to be periodically cleaned.

3. THE CONTEXT FOR THIS PROJECT

In Figure 1 we provide a logical framework for application of the SCIT approach. The figure shows two different SCIT-protected domains. The first is a mobile ad hoc network, of the kind that might be set up on the battlefield or a hostile or remote environment. The second is a SCIT domain within a more protected, wired enclave. SCIT protocols are run inside a single domain. Each domain continuously runs self-cleansing algorithms and protocols. This guarantees that the system is periodically immunized from successful, long time cyber attack.

We believe that the SCIT approach applies to a wide variety of military environments, including the static environments like a military hospital and small specialist offices and more mobile units that may be communicating using ad hoc wireless networks. We note that the overall security of the system depends on the weakest link. By ensuring that this link is periodically cleansed, SCIT can be used to improve security for networked static and mobile systems

Figure 1: Logical SCIT Topology



To highlight the potential impact of SCIT, we discuss below the application of SCIT in two typical application domains:

1. Small and mid size users. Specialists' offices and military field units are examples of such users. Such users have low Information Technology (IT) infrastructure investment, and the level of IT support is also low. These users have a heightened consciousness of the privacy requirements for their patients, but have limited expertise and resources for security monitoring and administration. The data in this environment is potentially stored on a single server. Our research for such users focuses on providing "security for dummies" technologies that relieve the above problems. In particular, we develop self-cleansing technologies for typical small-user setups to minimize the cost of security maintenance, or when successful intrusion occurs, to contain the damage. Our efforts in building a SCIT demo

platform have made great progress in one critical component of small-user environments, namely the firewall. Figure 2 compares a typical secure single server environment with SCIT based environment.

2. Large institutional users. Military hospitals are examples of such users. These users have significant IT investment, and regular in-house, on-site IT support. These users have higher availability of resources, and IT expertise is available. Generally, these organizations are conscious of the requirements of secure computing environment, and have resources for monitoring and maintaining the system security. In such environments, security has to be maintained at the level of the servers and at the network level. Our research for institutional users will concentrate on integrated, network-wide SCIT approaches. Of course, the scope and high level of integration in such an approach renders it significantly challenging. As a starter, we have studied existing network routing technologies and identified a type of routing protocols, called Link-State Routing (LSR), as "SCIT-compatible." We also notice that the most widely used "local" routing protocol in the Internet, namely, Open Shortest Path First (OSPF) protocol, uses link state routing. (A local routing protocol is used for traffic routing within individual institutions or organizations.) Figure 3 compares a traditional network based system with a SCIT based network system.

Figure 2: Single Server Secure System Architecture with SCIT

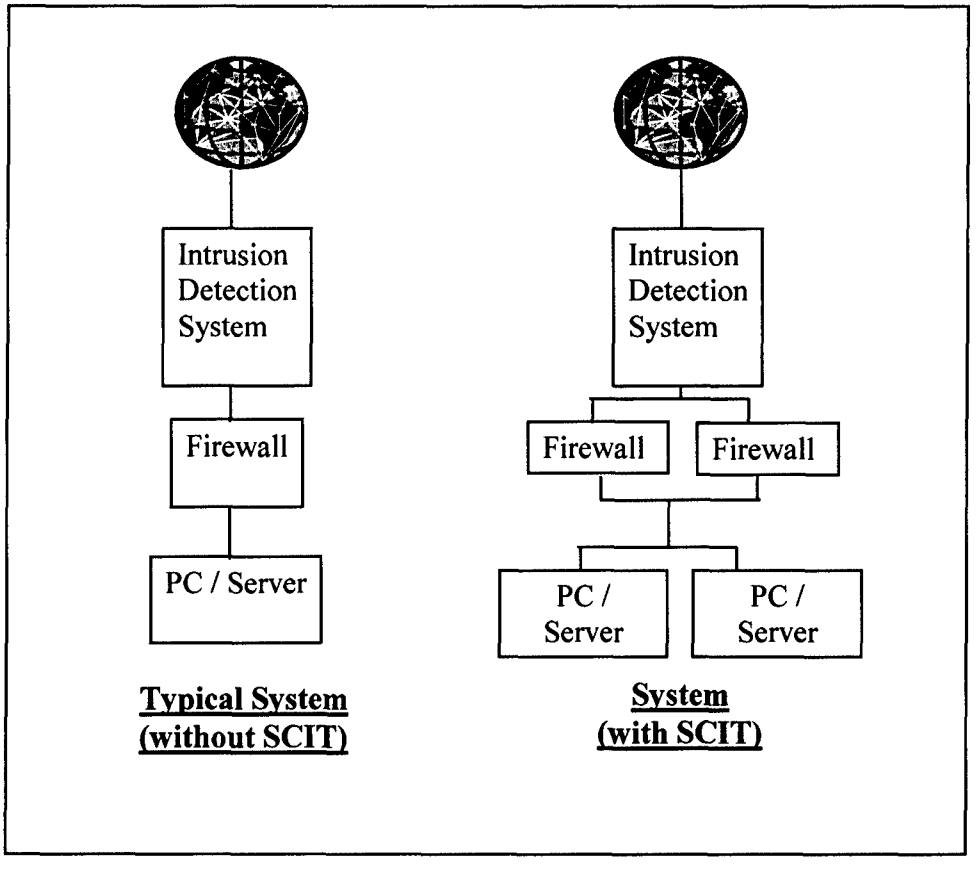
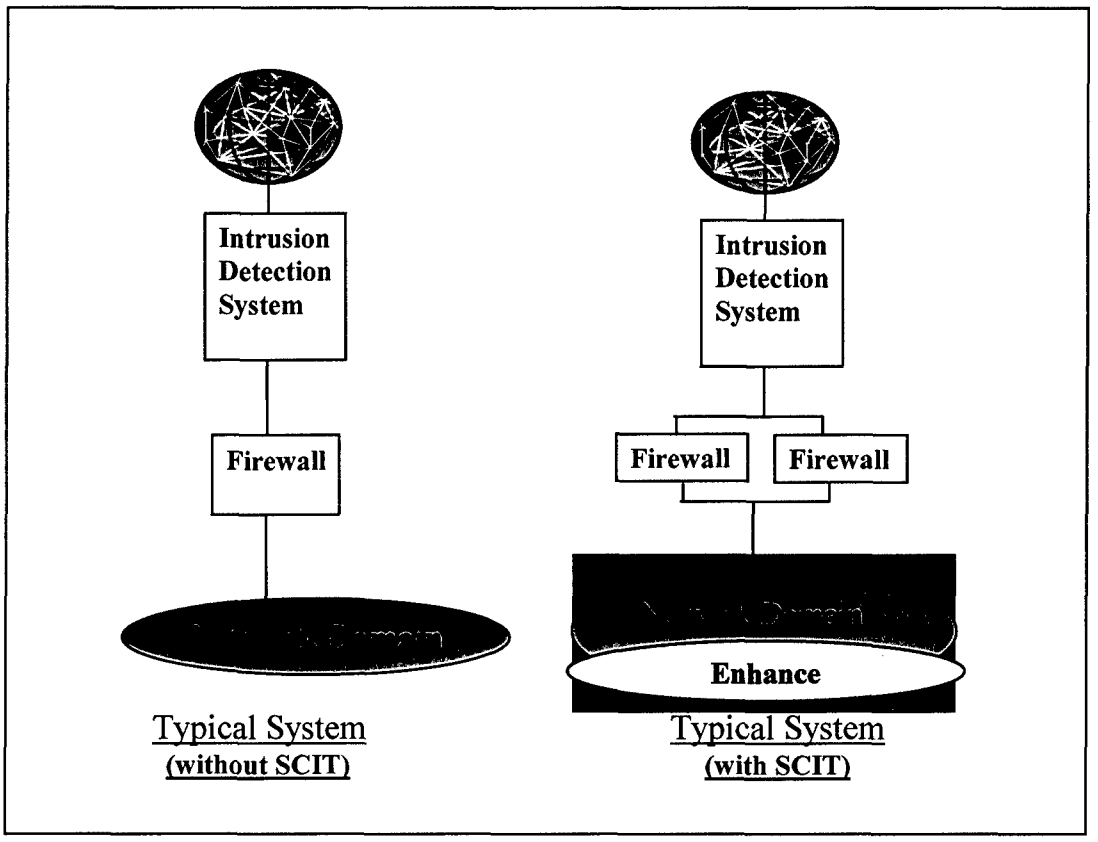


Figure 3: Secure Network Architecture with SCIT



4. INTRUSION CONTAINMENT

4.1 Frequency of Self Cleansing

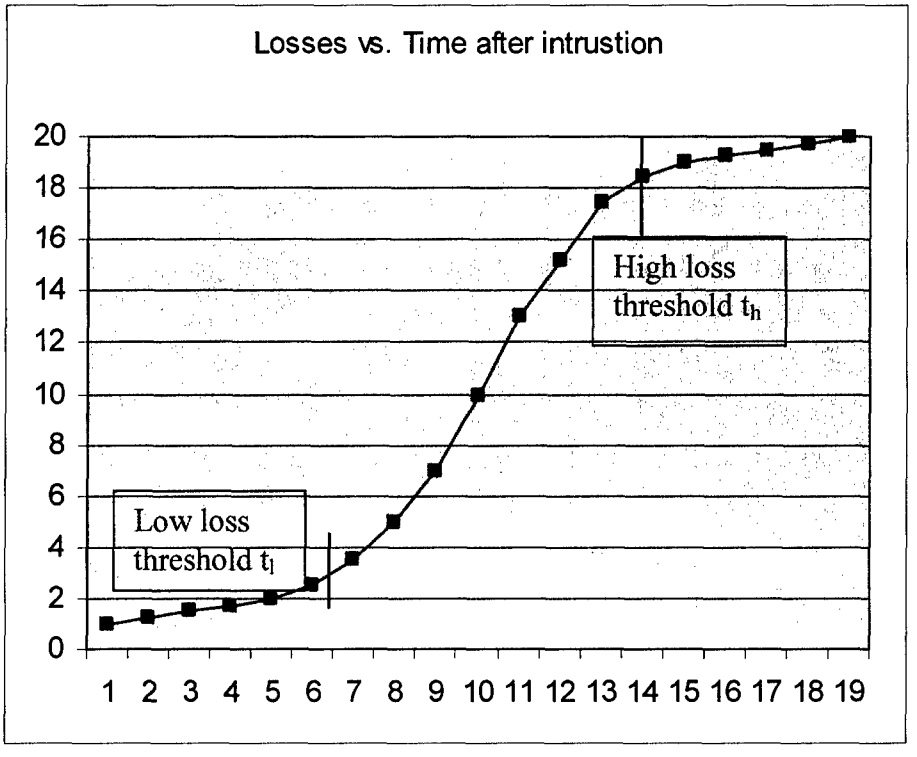
In spite of the best Firewalls and Intrusion Detections Systems, it is possible that intruders (hackers) will be able to launch a successful attack. Our zero-trust approach is geared towards this pessimistic scenario. Our approach of self-cleansing limits the amount of time that a successful intruder has to do "bad" things. More frequent self-cleansing operations will reduce the time the intruder has to do damage. However, very high frequencies of self-cleansing will either force high degree of hardware redundancy or reduce the system performance and also have the potential of reducing the quality of the presentation. Consequently, we ask the following question:

"What should be the frequency of self-cleansing?"

To answer this question, one needs to model the loss suffered with respect to the time that the intruder can spend in the system. For this purpose we define the Intruder Residence Time as the time that the intruder has to do damage in the server. We anticipate that the loss curve will be an S-curve of the form in Figure 4. If the Intruder Residence Time is less than the low loss threshold, then the cost of the intrusion is low, while an Intruder Residence Time greater that the high loss threshold will lead to near max loss. The steep slope between the two thresholds indicates that it is necessary to limit the Intruder Residence Time to less than low loss threshold.

In terms of the SCIT implementation, the time between consecutive self-cleaning events should be less than the low loss time. We note that the determination of the loss curve requires an empirical study and an assessment of the value of the data stored on the server.

Figure 4: Impact of Intruder Residence Time on Losses.



4.2 Authentication of Self Cleansing – SCIT Certificates

As we have argued, a significant advantage of self-cleansing is the ability for system users to be assured of a secure system. An important aspect of this is for users and system client to be able to verify that the self-cleansing has actually been performed. This verification requires SCIT-based *authentication.* From a security point-of-view, authentication is being able to prove who you are. Data integrity is proving that an object has not been changed or manipulated, expect by an authorized user. We strengthen this definition for SCIT. We say that SCIT-authentication is the ability to guarantee that a system object is available after a particular self-cleansing event. SCIT-defined authentication can be achieved via the introduction of a SCIT certificate.

Certificates are used in distributed systems to enable different users to exchange public security keys without having to directly contact a well-known security key provider. In effect, if I present you with a certificate, I can "prove" to you that I am who I claim I am, if you believe the authority that granted me the certificate. It is possible, using well-known cryptological techniques, to provide an unforgeable guarantee of a certificate signed by a trusted authority. We will produce a similar scheme for the SCIT environment.

Clients of SCIT services need SCIT certificates as unforgeable proof that an object has been cleansed. In this way, clients both inside and outside of a particular SCIT domain can receive the benefits of a SCIT-ized system. Notice that the certificate does *not* prove authenticity or data integrity for a client; rather, it proves to a client, which may be a human user, that an object has been self-cleansed.

There are several parts of this problem that need to be addressed. Here we present some of the questions that must be answered. First, which entity should produce a SCIT certificate? Second, what should the certificate actually contain? Third, which techniques can be used to minimize the overhead of producing and using the certificates? Finally, which clients might require the SCIT certificates, and how would it be used?

Our analysis assumes that there are some fully trusted entities when the system recovers. This helps answer the first question of which entities should produce a SCIT certificate – it is these trusted entities. One example is the multiple firewall server system shown in Figure 2.

The next question is what type of data needs to be in the certificate? Conceptually, the certificate needs to prove the identity of the sender, in this case the trusted self-cleanser. In general, proof of identity is an issue of the systems' *trust* model. This is currently considered an open area in security research, and later we describe some of the issues here. For purposes of our SCIT certificate, we assume that the identity of trusted SCIT server can be verified in one of two ways. The first way is if there is a shared secret key between the client and the trusted server. Recall that the clients' security and identity is not the issue here. If the client is in the SCIT domain then it will be self-cleansed. If it is outside the domain then the only purpose of the certificate is to prove that different objects have been self-cleansed. The second way of proving identity is via a public certificate, as proposed in X.509, which is currently the industrial standard.

The other fields in the SCIT certificate include an identifier for the cleansed object. For instance, if web server is currently cleansed via multiple copies, then the web server address can be identified. It is not necessary to explicitly identify which replicated copy is being used. Each SCIT certificate must also have an unforgeable way of identifying the current cleansed period, i.e., when the object was cleansed and for how long this should be.

The next question is what protocols should be used to distribute the SCIT certificate, and what cryptological approaches can be used to authenticate the information in the SCIT certificate. Delivery of the certificate does not present any special problems, and can be accomplished by whatever mechanisms are currently used by the system, for instance via a network level multicast (sending a single message to a group of receivers), by a bulk e-mail, etc. Authenticating the information in the certificate, including the interval of the cleanse cycle, can also be handled in several of ways. One way would be to simply encrypt the cycle number and timestamp within the message. Although this approach will work, one drawback is that it forces all of the receivers to perform a potentially computationally expense decryption for each certificate. This may be a burden for clients that have minimal computational and network resources, such as mobile nodes in a battlefield situation.

In order to alleviate the above problems we propose that the SCIT cleansing interval information is sent via multicast on a per object basis to all interested parties. Further, the information itself can be authenticated via a cryptological mechanism known as a *one-way* hashing function. It works as follows: all clients that are interested in a particular object receive SCIT certificates on a multicast group, e.g, by listening to a well-known network connection. The information that authenticates the value of the cleansed interval is obtained by using the one-way hash function on a particular SCIT field. The advantage of this scheme is several-fold. Only interested clients need to receive the SCIT certificate. One-way hash functions are known to be secure, so that as long as the identify of the SCIT server can be verified, the value of the SCIT interval can be authenticated as well. Further, one-way hash functions do not require that each client receive earlier SCIT-certificate messages. This means that messages can be lost or simply ignored, until the client requires a certificate for a particular object. In fact, we demonstrate below that this scheme does not even require that an entire certificate be sent each time an object is cleansed. This may be of particular advantage in resource poor or hostile environments.

An additional question is how the certificate could actually be used. Using the above approach, this is demonstrated by example. Assume that at some point a client will want to access a SCIT-ized web server. The client obtains the SCIT-certificate from the trusted SCIT server. This provides the necessary information, including some necessary values for the one-way hash function that the client will need later. When the client wishes to access the web server it also wants to verify that the server has recently been

cleansed. It does this by first listening to the appropriate network connection for the Web Server. Once it obtains the current certificate can verify the current cleansing cycle either through full-blow decryption or by use of the one-way hash function. In the later case it is possible to significantly reduce the amount of overhead required, because all that is required is the current value of the cleanse interval and a one-way hash value, which is used by the client to authenticate that the message was sent from the SCIT server. From a technical point of view, the input to the hash function is a combination of previously calculated hash values and the current cleanse interval.

Finally, we also specify the minimal requirements for the information contained in the SCIT Certificate are the *proof of identity* for the issuer, an identifier for the cleansed object, and timing interval relating for when the object was cleansed and, optionally, how long this particular cleansing cycle will last. In distributed systems, proof of identity is described in terms of the application's trust model. As indicated in our earlier study, the trusted SCIT certificate server can be verified in one of several ways. One way is to have a shared secret key between the client and the trusted server. Let us elaborate on this model. Shared secret keys can be deployed in a variety of ways, but typically this approach is most appropriate for small and medium sized enterprises. By using the shared secret key approach, each time an object is cleansed the SCIT Certificate Server produces a new certificate containing the cleansed objects identifier, the time that it was cleansed, along with other potential information such as for how long before this cleansed object will be cleansed again.

Using the shared secret key technique, the validity of the certificate is obtained in one of several ways. One was is to encrypt the entire certificate using the shared secret key. In this way, only trusted clients could decrypt the key and therefore obtain the information. Another approach is to send out certificate information in unencrypted form, but to digitally sign the certificate using the shared secret key. In this approach a client will be able to prove that the certifier is who they claim to be, because correct signing is possible only if the signer has the key.

The shared secret key approach is powerful for most small systems that do not have too many objects to manage or users to support. However, there may be important performance drawbacks to this approach as the system scales up in size. For instance, if there are many users and SCIT certificate servers then there will need to be many pairs of shared keys. A more general approach is to use existing trust architectures and to *fold* the management, protocols and trust model of the SCIT certificate into current proposed public certificate standards. We now explain how this could be done using X.509 [X509], one of the more popular standards. It should be clear that the traditional use of X.509 is for *user* authentication. However, as we will see below, the standard is flexible enough so that we can directly apply it to SCIT.

X.509 defines an entire framework for authentication services. Part of the X.509 standard is the definition of a common, general-purpose certificate. This certificate format is currently used in a number of important Internet protocols, including S/MIME, for e-mail, IP Security, for general network and transport layer security, and SSL/TLS, for secure distributed programming. X .509 does not specify the exact security algorithms to use; rather, it defines the data format and trust model for their use. Several features of the X.509 certificate format make it an attractive possibility for SCIT. First, it is an existing standard, reasonably well-understand and therefore will be compatible with other approaches. Second, it used Public Key Cryptography, and therefore avoids the problem of shared secret key and the attendant key distribution problem.

The data fields in the X.509 certificate are appropriate in a SCIT Environment. We are basing our discussion on X.509 version 3. This version includes a very flexible naming and directory service conventions, so that certificates can be directly used for both SCIT objects and users. Version 3 has specifically the ability for the user to define an entire name space of managed objects. Another data field is the period of validity. This consists of two dates, the first and the last on which the certificate is valid. This can be used to indicate the cleansing time of the object. Finally, X.509 allows the use of user-defined policy constraints that can be used to manage the scope and usage of SCIT certificates.

Based on the above discussion we believe that the functionality of the SCIT certificate can be achieved using an X.509-like mechanism. Notice that this approach addresses the problems of scalability and interoperability with other security techniques. It also allows the use of the SCIT certificate both inside and outside the SCIT domain. Finally, the certificates themselves can be encrypted, so that if required cleansing information can be kept confidential and secure.

[X509] See http://www.ietf.org/html.charters/pkix-charter.html for a comprehensive web site with pointers to the X.509 standard.

4.3 Trust in SCIT

Although we believe that the SCIT certificate provides a mechanism for authenticating self-cleansing actions, one related issue that has not been addressed is trust. Just as in human society, trust is not a boolean property; it comes in an infinite number of degrees. One of our tasks is to determine what an appropriate level of trust is for the system and how to establish it. Consider the following. In a certificate based system, networks or chains of certificates are built to establish identities. This is straightforward when taking a global (system-level) perspective; however, from a local node level, things get more complicated. Suppose a node A knows it can trust certificates from another node B. When A receives a certificate from node C who claims to has been certified by trusted B, A must determine whether or not to believe this claim. This requires A to consider the certificate chain back to B. There is a large body of work in trust management that we hope to be able to use in this context. In addition to approaches designed to work in certificate-based systems, there has been recent work on trust in credential-based systems. This may be of particular interest in tele-medicine systems because it merges the issues of authentication and authorization.

4.4 Software-Level SCIT

The goal of this research was to investigate the feasibility of a self-cleansing approach that forced hardware to re-boot at intervals in order to thwart intrusions. However, this approach is not complete without considering both how to cleanse the software and also how software running on these systems can be made to react appropriately when this self-cleansing occurs. Both of these problems require some method of making runtime changes to a (possibly distributed) software application.

Dynamic reconfiguration is the process of making changes to a running application. These changes can include moving parts of a distributed application to other platforms, replacing components of the application with updated components and even changing the overall configuration (components and connections) of the application. Techniques for dynamic reconfiguration have been studied in other contexts, such as specialized load balancing; we believe some of this research is useful in the context of self-cleansing. In particular, heterogeneous process migration based approaches may be used to continue to provide access to software systems as the underlying hardware is being cleansed. It may also be the case that techniques for component updating can be used to cleanse software if the running component itself is corrupted.

Our current cleansing techniques borrow ideas from fault tolerance based approaches, the use of multiple copies of a system element so that availability is not compromised while an element is being cleansed. However, even given this structure, there are two different possible dynamic approaches to consider: proactive and reactive. Reactive reconfiguration attempts to respond to changes in the current situation. The Survivability architecture work at UVA uses explicit control to manage information systems using data from infrastructures, their information systems, and their operational environments. Their approach is to find a way to continuously make configuration decisions based on the current state of the overall system. Their approach is decentralized and hierarchical.

While we can learn something from this reactive approach, it assumes that intrusions are detectable at some level, possibly by a change to the environment. If we assume that some intrusions are not easily detectable and that we would like to thwart intruders before damage is done, then a proactive approach that works in conjunction with the underlying system cleansing could be important. In proactive reconfiguration, configuration changes are driven by the need to prevent a possible intruder from harming the application and its data.

A number of issues arise when considering using dynamic reconfiguration techniques within the context of SCIT:

- How to ensure that dynamic changes to the application configuration do not change the application's functionality?

Consistency issues arise in the context of dynamic changes to an application configuration. For example, when cleansing requires switching between primary copies of a software component that is involved in a transaction, either the switch must wait until the transaction has ended, or the switch must be done in a manner that hands the state of the transaction over to the new component.

- How to ensure that the software still meets availability requirements?

Techniques for adding self-cleansing to overall systems must have a minimal impact on the availability of the system. This is not a solved problem in the context of dynamic software reconfiguration; some techniques require waiting for a given state before changes can be made to the configuration of the software. This waiting time may or may not be unbounded.

- How to make existing applications adaptive?

We will want to consider what requirements we want to place on components in this context. If we require components to be able to actively participate in proactive reconfiguration changes, we may be unable to address applications that use COTs components. On the other side, if no application components can participate actively, this may limit the types of configuration changes we can consider. Some middle ground may be the answer -- either where some of the components can participate or where we can wrap components (or use "agent" components) to deal with the reconfiguration.

- How to secure the mechanism itself?

Adding reconfiguration mechanisms to an existing environment gives an intruder an additional, possibly very tempting, target. Any approach that increases system vulnerabilities must be discarded.

References:

Survivability Architectures: Issues and Approaches,
J. Knight, K. Sullivan, M. Elder, C. Wang,
DARPA Information Survivability Conference and Exposition, 2000.

Information Survivability Control Systems,
K. Sullivan, J. Knight, X. Du, S. Geist
Proceedings of the 21st International Conference on Software Engineering, IEEE, 1999, pp. 49-71.

5. CURRENT STATUS OF THE DEMO

In this project we demonstrated the application of SCIT to a significant component of a traditional secure system. At this point of our development, we are able to show some of these features. In this note, we describe the architecture, the current status and future plans.

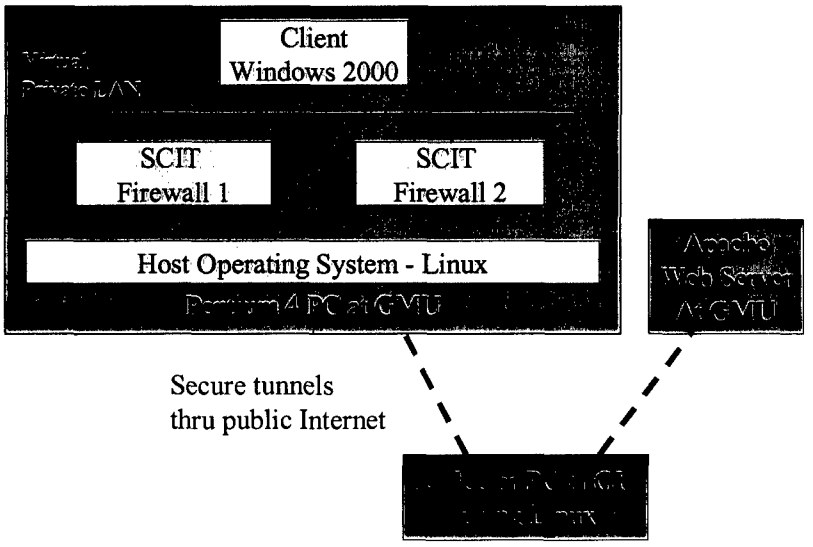
5.1 Objective of the Demo

Assess the impact of SCIT firewall on system performance. This assessment will be conducted at two levels – perceptual and packet loss assessment.

5.2 SCIT Firewall Demo Architecture

The overall architecture is shown below.

Figure 5: SCIT Firewall Demo Architecture



A secure tunnel connects an Apache web server at GMU, a reflector at GU, and a client (Windows 2000) inside the SCIT firewall. A client uses the Explorer on Windows 2000 to connect to the Apache web server. The client Windows, protected by SCIT firewalls, reaches the sever via a secure tunnel that goes from the client machine to a reflector machine . This setup is designed to emulate a remote client accessing a private server using VPN (virtual private networks).

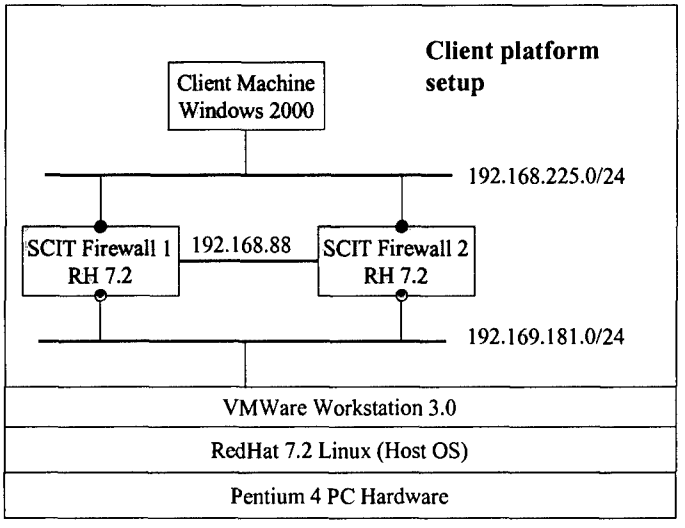
5.3 Client Platform

The client Windows machine and its SCIT firewalls are implemented as *virtual machines,* on a Pentium 4 PC running RedHat 7.2 Linux. This is achieved by installing on the Pentium 4 PC the VMWare Workstation 3.0 for Linux, which enables the use of one or more *guest operating systems*, namely virtual machines, on top of a host system. VMWare 3.0 supports not only virtual machines but also *virtual networks*, switched Ethernet networks emulated by the VMWare 3.0 software. We use this feature to build two virtual networks, one with subnet ID 192.168.181.0/24 and the other 192.168.202.0/24; please see Figure 6. The first subnet connects the host to the two firewalls. Both firewalls are implemented by a specialized version of RedHat 7.2 Linux. The latter subnet connects the firewall boxes to the client Windows. Inbound traffic are first received by the host (host interface not shown in the figure) and relayed to one of the firewalls, which filter and relays the traffic to the client Windows. A third subnet, 192.168.200.0/24, is used by the two firewalls to probe each other.

5.4 SCIT Firewall Implementation

In the figure below, two sets of network interfaces are highlighted by colors. The green ones are firewall interfaces connecting to the host. Because the SCIT firewalls operate alternatively, the host sees only one firewall at any moment of time, or more precisely one firewall interface. Thus, the two green interfaces must share one IP address. This "green" address is the gateway address in the Host routing table to reach subnet 192.168.225.0/24. Likewise, the red interfaces are the SCIT firewall interfaces that connect to the client Windows and must share one IP address. This "red" address is configured as the gateway address on the client Windows. Now consider a time when Firewall 1 is operating and Firewall 2 has just finished self-cleansing and is about to take over. Since Firewall 1 presently owns both the green and red addresses, Firewall 2 needs to "grab" the two addresses from Firewall 1. We achieved this by Gratuitous ARP messages (ARP stands for address resolution protocol).

Figure 6: Client Platform Configuration using Virtual Machine



To implement firewall rules, we use the kernel-based firewall implementation provided by Linux, called ipchains. The operations of each firewall are controlled by a shell script that executes the following steps. In the script, the first step is executed immediately after the underlying firewall has completed its rebooting.

1. Configure firewall rules. Start traffic filtering and relaying in the background. The two tasks will continue to be performed in the background until the machine is shutdown for rebooting in Step 5.
2. Broadcast on subnets 192.168.225.0/24 and 192.168.181.0/24 Gratuitous ARP messages to announce the ownership of firewall IP addresses. This step lasts 10 seconds; one ARP message per second.
3. Wait for 10 seconds. This delay gives the other firewall extra times to detect the activities of this firewall and to reboot.
4. This firewall now assumes the other firewall is in the process of rebooting. It sends ping message periodically over the 192.168.88.0 network to probe the other firewall, until a reply message is received. The receipt of a reply indicates the other firewall has completed self-cleansing and is ready to take over.
5. Reboot (and thus return to step 1 after completion).

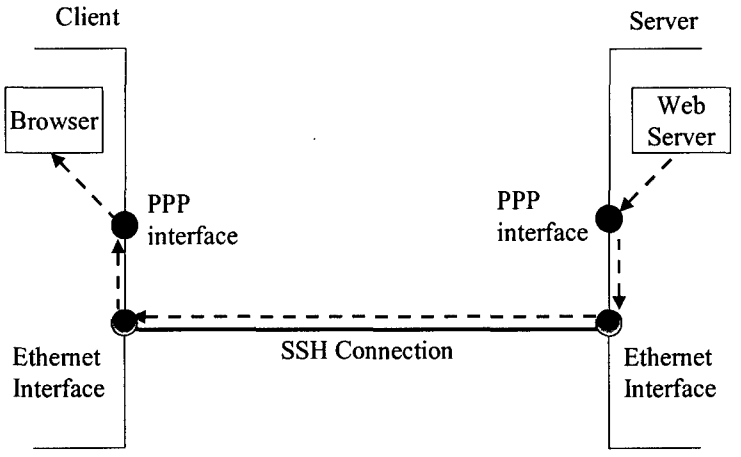
5.5 Secure tunneling

Although tunneling is not a SCIT technology, it is widely used in virtual private networks (VPN) and its compatibility with SCIT firewalls is important. In our demonstration, a secure tunnel is built between the server and client through the public Internet to reflect this practice.

While there are many commercial VPN products on the market, we choose an open-source VPN implementation to save costs. As seen in Figure 7, our secure tunnel uses the PPP (Point-to-Point Protocol) over an SSH (secure shell) connection. From the client's viewpoint, the tunnel is like a PPP-controlled modem connection to the server (without the bandwidth limitation of a modem), where the "modem line" is emulated by an SSH connection through the public Internet. The SSH connection connects the Ethernet interface cards on the client and server machines. The PPP protocol builds two PPP network interfaces at the end points of the SSH connection. The two PPP interfaces are sometimes called *virtual* for these do not correspond to real networking devices (this does not use and has nothing to do with the VMware software). User-level applications use PPP network interfaces to communicate. In Figure 7, the web server application sends to the client a web page through the PPP interface on the server machine. The page is

then encrypted by SSH (still on the server) and leaves the server machine via the real, Ethernet interface card. When the page arrives at the Ethernet card on the client machine, the page is decrypted and relayed to the PPP interface. The client application, a browser, retrieves the page from the PPP interface and renders the page on the screen. We point out that the arrows in the figure are used to emphasize the direction of transmissions in the above example. The secure tunnel supports transmissions in two directions.

Figure 7: Securing Tunneling



5.6 Current demo results

From the client inside the SCIT firewall, the user opens a browser window and begins web surfing. We open two additional windows indicating which of the firewalls is operational. In the screen capture shown in Figure 8, these correspond to the black background windows on the top of the screen. The right-upper window shows that the presently running firewall is probing (unsuccessfully) the other firewall. That is, the firewall is executing the Step 4 given above. The left-upper window shows the booting message of the second firewall. The TATRC home page at the bottom is displayed by the client Windows 2000. The page is, of course, obtained through the SCIT firewalls.

In general, for a typical HTML encoded web page, a firewall switch is barely perceptible – examination of the trace sometimes shows occasional losses of packets, but it appears that the retransmission of the packet is fast enough that the user cannot perceive the difference. Indeed, the setup is good enough for our own production uses. While browsing the web to research the secure tunnel technology, our RA regularly uses the Explorer on the client Windows 2000 with the SCIT firewalls running. With the setup, we observed self-cleansing cycles in the vicinity of 90 seconds.

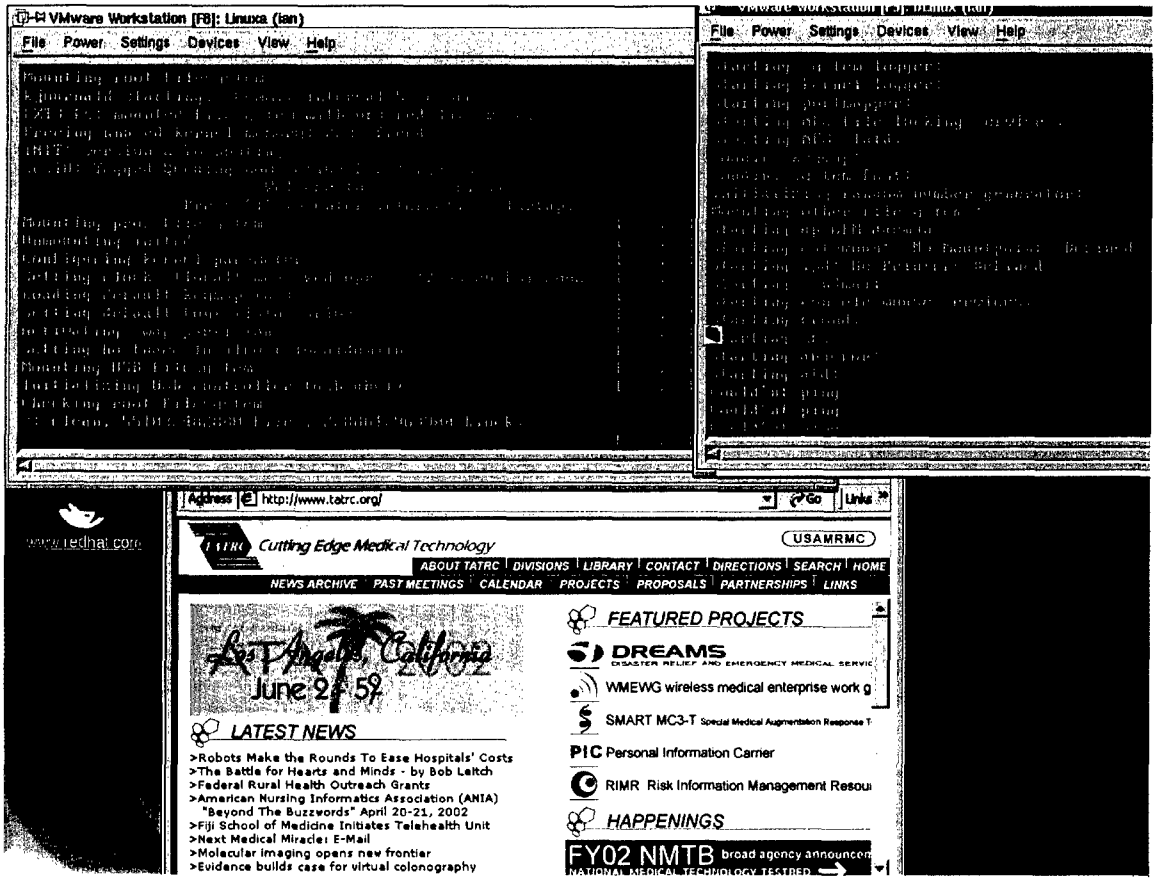
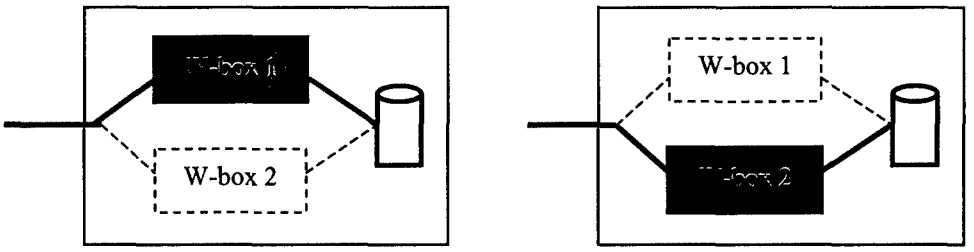


Figure 8: Typical Screen Layout to Simultaneous Monitoring of the SCIT firewall and Web Surfing

5.7 SCIT Web Servers

Our second prototype is a SCIT web server based on the concept depicted in Figure 9. As in the case of the firewall prototype, two boxes are used build one SCIT web server. Each server box is a complete computing device, including local storage devices for the operation system and application programs. The two boxes have access to a shared storage to store contents of web pages. When one box is working, the other server is performing self-cleansing by rebooting itself, followed by integrity checking of critical system files and static web contents.



(a) w-box 1 in operation (b) W-box 2 in operation

Figure 9. SCIT web server concept

The SCIT web server testbed is also based on the Virtual Machine software from VMWare, Inc. [6]. In the testbed, the two w-boxes are virtual machines running RedHat 7.2 Linux. The shared storage is provided by an NFS server running on a RedHat 7.2 virtual machine. The underlying host machine is a Pentium 4 PC also running RedHat 7.2 Linux. As seen in Figure 9, we build two virtual networks, one with subnet ID 192.168.181.0/24 and the other 192.168.225.0/24. A third subnet, 192.168.88.0/24, is used by the two firewalls to probe each other. In this prototype, critical system files and web contents are protected by digital signatures. While digitally signing critical information is not new, we further enhance its effectiveness by integrating the signing tasks with self-cleansing cycles, given below.

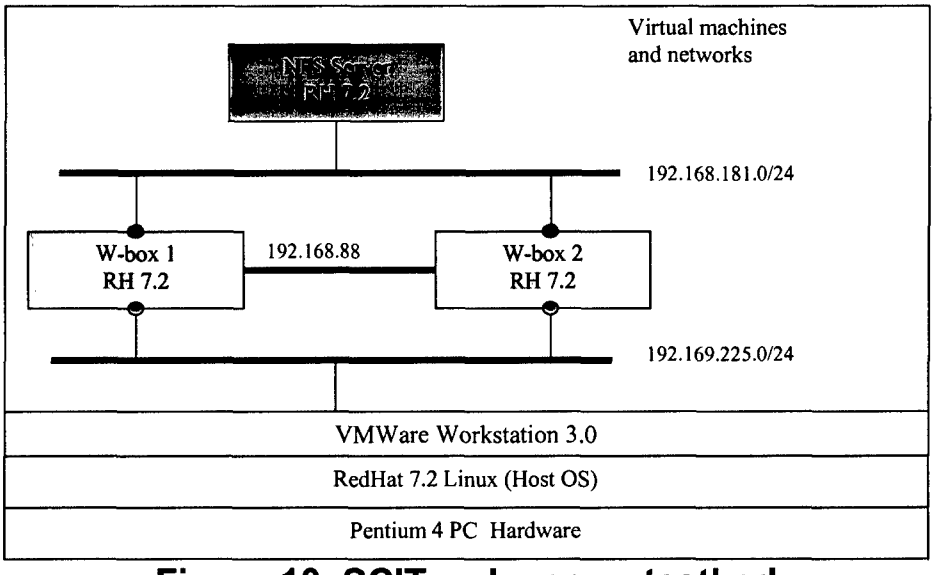


Figure 10. SCIT web server testbed

1. Check signature integrity, using previous keys.
2. Generate new keys and re-sign web pages with new keys.
3. Claim the web IP address by the VRRP (Virtual Router Redundancy Protocal).
4. Probe the other box, until a response is received.
5. Reboot and return to step 1.

In our SCIT server, new signatures are produced for protected files using new keys in every self-cleansing cycle. The first task of a sever box after rebooting is to check signatures produced in the previous cycle using the old keys and subsequently produces a set of new keys to compute new signatures. In this way,

the enemy has only the time window of one cycle to break/steal the current key. Even if the enemy succeeds in the task, the time window to inflict damages to the system is limited to one cleansing cycle.

Notice that we depart from the firewall prototype in the technique of claiming a shared IP address. In stead of using Gratuitous ARP, the VRRP (Virtual Router Redundancy Protocol) is used. Originally designed for the failover of the gateway router in a subnet, the protocol can be used for any server to share an IP address with its backup and perform smooth IP address handover in case of server failures.

5.8 Discussion

Our SCIT prototypes demonstrate the effectiveness of SCIT in several important aspects. First, system self-cleansing limits the amount of time that a successful intruder has to stay in the system and inflict damages. The longer this *Intruder Residence Time* the greater the damage and loss. We anticipate that the loss curve will be an S-curve of the form in Figure 4. If the Intruder Residence Time is less than the low loss threshold, then the cost of the intrusion is low, while an Intruder Residence Time greater that the high loss threshold will lead to near max loss. The steep slope between the two thresholds indicates that it is necessary to limit the Intruder Residence Time to less than low loss threshold. The low loss threshold reflects the reality that it takes a certain time window for a hacker to be able to issue malicious commands, exploit backdoors, install Trojan horse programs, and/or steal/destroy data. A conservative estimate of the low loss threshold is in the range of minutes. Although there is no hard data for building the loss curve, there are reports that can help the process of building such a curve. For example, in [5] it is reported that in the context of on-line banking, security experts believe that a theft of $5,000 to $10,000 can be carried out over a few weeks, while larger losses up to $1 million are likely to take four to six months. In this context it is emphasized that the loss curve must account for the possibility that the *Intruder Residence Time* is spread over more than one successful breach of the system. In our experiences, self-cleasnsing cycles are in the range from tens of seconds to minutes. Such short cycles limit the Intruder Residence Times and thus severely limit the damages of even successful intrusions.

Second, combining SCIT with traditional system defenses enhances both. In the SCIT web server prototype, the effectiveness of digital signatures is strengthened by the use of new keys in every cleansing cycle. In the meantime, the effectiveness of self-cleansing is enhanced by the integrity checking of signatures and accompanying recovery procedures when the check fails. The result is very short time window to defeat *two* defenses. We believe SCIT has great potentials in collaborating with other traditional system defenses, such authentication, certification, access control, checkpoints, rollback procedures, and so forth.

Third, through our experiences of the two SCIT prototypes, we notice a close relation between SCIT a branch of computing industry, namely, *high-availability computing*. A high-availability system typically uses backup systems to ensure continued customer services in face of primary server failures. SCIT servers share many design challenges with high-availability systems, such as the seamless takeover by the backup, sharing of IP addresses between the primary and backup, and monitoring the primary servers. Indeed SCIT can be considered as an extension to high-availability systems in the sense that artificial failure events are introduced to force periodic takeover of the backup server and the self-cleansing of the primary server. This view of SCIT justifies its technical feasibility: every high-availability product on the market gives confidence to a corresponding SCIT system following similar designs. Indeed we already used open-source packages from the High-Availability Linux project in our SCIT prototypes. Examples of existing high-availability systems include web servers, NFS servers, authentication services, firewalls, and IPsec gateways.

Fourth, SCIT can also be considered as a resilient technology. A resilient technology in system defense automatically slows attacks so buying time for containment but does not root out the attacks themselves [Will02]. The virus throttling technique proposed by Matthew Williamson for instance slows down virus propagation by introducing delays to those connections that likely carry viruses in payloads (the distinction can be made by observing connection destinations --- an infected server typically connects to as many destinations as possible to spread viruses while a normal server makes connections to a much smaller set of

destinations). A second example is to introduce system call delays to applications not in conformance with "typical" system call behaviors (and thus likely compromised). With such technologies in mind, consider our SCIT web server prototype. If the enemy does not have the capacity to break a new key in each self-cleansing cycle, then SCIT succeeds in fending off the attack. Otherwise, SCIT at least slows the attack by forcing the repeat of some attack procedures after every rebooting. Our prototypes in this light demonstrate two benefits of SCIT: an effective SCIT technology constitutes a defense; a less successful one degenerates to a resilient technology and still contributes to the overall security of the system.

6. CONCLUSIONS

In this research we have built the SCIT firewall and a class of SCIT web server. We conducted tests in the laboratory to verify that the switch between SCIT boxes did not result in performance degradation for the user. T he experience from this project and the firewall and web server tests have shown that the SCIT concept can be further generalized. This validation of the SCIT has lead to our exploring the extension of this concept to more complex environments. Our future research in the application of SCIT will involve developing SCIT for single processor systems that are more complex than the SCIT firewall or web server, and analyzing SCIT in the context of network of processors as compared to a stand alone system. For example we expect that SCIT approach can be applied to protect NFS Server, Certification Server or Certification Client.

References

[ACDK96] H. Abu-Amara, B. A. Coan, S. Dolev, and A. Kanevsky, "Self-stabilizing topology maintenance protocols for high-speed networks," *IEEE/ACM Trans. on Networking*, Vol. 4, No. 6, pp. 902 – 912, December 1996.

[AFVa95] D. Anderson, T. Frivold and A. Valdes, "Next generation intrusion detection expert system (NIDES)," RT SRI CSL-95-07, SRI International, Menlo Park, CA 94025-3493, May 1995.

[BGJo99] Joffroy Beauquier, Maria Gradinariu and Colette Johnen, "Memory space requirements for self-stabilizing leader election protocols", *Proceedings of the 8th annual ACM Symposium on Principles of Distributed Computing*, pp. 199 – 207, May 1999, Atlanta, GA USA.

[Bish99] M. Bishop, "Vulnerabilities Analysis," *Recent Advances in Intrusion Detection* (Sep. 1999)

[BRSN90] P. Banerjee, J. T. Rahmeh, C. Stunkel, V. S. Nair, K. Roy, V. Balasubramanian, and J. A. Abraham, "Algorithm-based fault tolerance on a hypercube multiprocessor," IEEE Trans. on Computers, Vol. 39, No. 9, pp. 1132 – 1144, September 1990.

[BuPa89] James E. Burns and Jan Pachl, "Uniform self-stabilizing rings," ACM Trans. On Programming Languages and Systems, Vol. 11, No. 2, pp. 330 – 344, April 1898.

[CNSW97] A. Chowdhary, L. Nicklas, S. Setia, and E. White, "Dynamic space sharing on clusters of non-dedicated workstations," in *17th International Conference on Distributed Computing Systems*, May 1997.

[CIDF] B. Tung, "Common intrusion detection framework (CIDF)." URL http://www.isi.edu/gost/cidf/, September 1999.

[Dijk74] E. W. Dijkstra, "Self-stabilizing systems in spite of distributed control," *Comm. ACM*, vol. 17, no. 11, 1974.

[HaRe83] T. Haerder and A. Reuter, "Principles of transaction oriented database recovery -- a taxonomy," *ACM Comput. Surv.*, vol. 15, December 1983.

[HoPu93] C. Hofmeister and J. Purtilo, "Dynamic reconfiguration in distributed systems: adapting software modules for replacement," in *13th International Conference in Distributed Computing Systems*, pp. 101 – 110, 1993.

[HoWhPu93] Hofmeister, C., White, E. and Purtilo, J. 1993. Surgeon: a packager for dynamically reconfigurable distributed applications, IEE Software Engineering, 8,2, pp. 95-101.

[Ho93] Hofmeister, C. Dynamic reconfiguration of Distributed Applications. Ph.D. Thesis, University of Maryland, College Park, 1993.

[HuMc99] Yih Huang and P. K. McKinley, "Tree-based link-state routing in the presence of routing information corruption," *Proceedings of IEEE ICCCN '99*, Boston, 1999.

[HuSo02] Yih Huang and Arun Sood, "Self-Cleansing Systems for Intrusion Containment," *Proceedings of Workshop on Self-Healing, Adaptive, and Self-Managed Systems (SHAMAN)*, New York City, June 2002.

[KrMa90] Kramer, J. and Magee, J. 1990. The Evolving Philosophers Problem: Dynamic Change Management. IEEE Transactions on Software Engineering, 15,11, pp. 1293--1306.

[Linux-HA] High-Availability Linux Project. Home page http://linux-ha.org/.

[LiPo99] U. Lindqvist and P. Porras, "Detecting computer and network misuse through the production-based expert system toolset," *IEEE Symposium on Security and Privacy*, Oakland, CA. May 9-12, 1999.

[MHPS92] C. Mohan, D. Haderle, B. Lindsay, H. Pirahesh, and P. Schwarz, "ARIES: A transaction recovery method supporting fine-granularity locking and partial rollbacks using write-ahead logging," *ACM Transactions on Database Systems*, vol. 17, pp. 94 – 162, March 1992.

[MoLO86] C. Mohan, B. Lindsay, and R. Obermarck, "Transaction management in the {R}* distributed database management system," *ACM Transactions on Database Systems}*, vol. 11, pp. 378 – 396, December 1986.

[MWill02] Matthew M. Williamson, "Throttling Viruses: Restricting Propagation to defeat Malicious Mobile Code," in Proceedings of 18th Annual Computer Security Applications Conferences, (Las Vegas, Nevada), December 2002.

[NSTh02] D. Newman, J. Snyder, and R. Thayer, "Crying wolf: False alarms hide attacks." Appeared on Network World Fusion at http://www.nwfusion.com/techinsider/2002/0624security1.html, June 2002.

[Perl88] R. Perlman, *Network Layer Protocols with Byzantine Robustness*. PhD thesis, MIT, October 1988.

[RFC2154] S. Murphy, M. Badger, and B. Wellington, "OSPF with digital signatures." Internet RFC 2154, June 1997.

[RFC2338] S. Knight, D. Weaver, D. Whipple, R. Hinden, D. Mitzel, P. Hunt, P. Higginson, M. Shand, A. Lindem, "Virtual Router Redundancy Protocol," RFC 2338, April 1998.

[RFC2385] A. Heffernan, "Protection of BGP sessions via the TCP MD5 signature option." Internet RFC 2385, August 1998.

[Spin97] John M. Spinelli, "Self-stabilizing sliding window ARQ protocols," *IEEE/ACM Trans. on Networking*, Vol. 5, No. 2, pp. 245 – 254, April 1997.

[SoFo00] A. Somayaji and S. Forrest, "Automated response using system-call delays," in Proceedings of the 9th USENIX Security Symposium, (Denver, CO), pp. 185--197, August 2000.

[Vmware] VMware Inc. Home page http://www.vmware.com/.

[Weyg96] Peter S. Weygant, *Clusters for High Availability*, Prentice Hall, 1996.

[Will02] Mathew M. Williamson, "Resilient Infrastructure for Network Security." HP Labs Technical Reports HPL-2002-273. Available at http://www.hpl.hp.com/techreports/2002/HPL-2002-273.html.

Self-Cleansing Systems for Intrusion Containment

Yih Huang
Computer Science Department
George Mason University
Fairfax, VA 22030
(703) 993 1540
huangyih@cs.gmu.edu

Arun Sood
Computer Science Department
George Mason University
Fairfax, VA 22030
(703) 993 1524
asood@cs.gmu.edu

ABSTRACT

In this paper, we discuss the application of high-availability computing systems to intrusion containment. Intrusion Management Systems (IMS) serve to protect complex computer systems from unauthorized intrusions. The traditional IMS approaches rely on intrusion prevention and detection, followed by implementation of intrusion resistance procedures. A key assumption of a traditional IMS is that it is possible to detect all intrusions. We believe that the sophistication and rapid evolution of information warfare require the more pessimistic assumption that undetected intrusions will occur and must be guarded against as well.

Our approach, called *Self-Cleansing Intrusion Tolerance* (SCIT), pushes the concept of high-availability computing one step further. In a SCIT system, a server is periodically assumed to have "failed," namely, comprised by undetected intrusion. Consequently, the server is brought off-line for cleansing and integrity checking while a backup takes over. Indeed, it is more appropriate to see a SCIT system as two mirror servers working alternatively than as a primary server and its backup. In this paper, we define the concept of SCIT, present our experiences in building a SCIT firewall prototype, and discuss the future work in more advanced SCIT servers.

Keywords

high-availability computing, computer security, intrusion containment, self-cleansing systems.

1. INTRODUCTION

Computer systems are becoming more complex and are increasingly vulnerable to cyber warfare. Typical (traditional) Intrusion Management Systems (IMS) are based on intrusion prevention and detection followed by implementation of intrusion resistance procedures [1,2]. The latter generally includes intrusion tracking and subsystem isolation. Such an IMS approach relies heavily on the ability to detect intrusion events in the first place.

We however make the pessimistic but realistic assumption that not all intrusion activities can be detected and blocked (at least not in a timely manner to avoid significant damage) and seek technologies to build secure systems which constantly assume the compromise of the system and perform self-cleansing, regardless of whether intrusion alarms actually occur or not. We argue that such an assumption is appropriate given the sophistication and rapid evolution of information warfare. It is especially important for critical distributed computing systems: To achieve the highest level of security, we must not be overconfident in either our knowledge of enemy tactics and technologies or our capability to fend off *all* attacks.

One implementation of self-cleansing involves rebooting the subsystem from a trusted storage device followed by, if necessary, system recovery, checkpoint, rollback, and data integrity checking routines. (A trusted storage device can be either a read-only storage device or any nonvolatile storage where information is cryptographically signed.) System availability is achieved by means of redundancy, that is, a second mirror system is brought online to provide services. In this way, SCIT can be considered as a branch of high-availability computing [3,4]: In a highly available system, sufficient hardware redundancy is built into the system so that a backup can immediately replace a failed system. In SCIT, the switching from one system to its mirror is not only triggered by failures; it is a regular routine designed to root out undetected intrusion activities.

To illustrate, we apply the SCIT approach to firewalls. Here we assume that the decision of whether to drop a packet is strictly made on a per-packet basis.[1] Firewalls are widely used to block undesirable, potentially hostile packets at the entry to a secure site. A successful and unnoticed firewall subversion will leave the door to the site open, exposing the internals of the victimized site to the outside

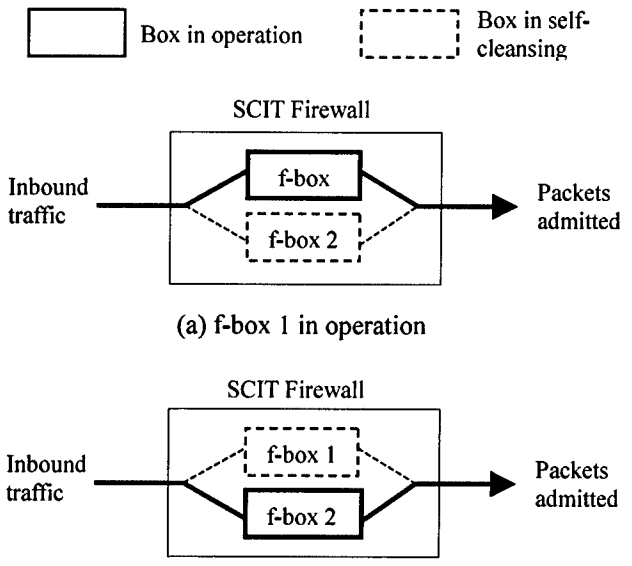
This research is supported by US Army's Telemedicine and Advanced Technology Research Center (TATRC), under contract number DAMD 17-01-1-0825.

[1] The cases of *stateful* firewalls, which maintain state information of ongoing TCP connections, will be more involved. Also, we do not consider proxy servers as part of the firewall.

world. As seen in Figure 1, we simply use two identical firewalls, called *f-boxes* in the figure. When one f-box is working, the other box will be performing self-cleansing by rebooting itself. Assuming that rebooting is from read-only devices, a rebooted f-box will be in a clean state and can perform packet filtering needed to protect the site. As such, even if the enemy managed to break into one f-box, its control over that box is limited to one self-cleansing cycle. More complicated SCIT systems will be discussed later.

Figure 1. SCIT Firewall

System self-cleansing limits the amount of time that a successful intruder has to stay in the system and inflict



damages. The longer this *Intruder Residence Time* the greater the damage and loss. We anticipate that the loss curve will be an S-curve of the form in Figure 2. If the Intruder Residence Time is less than the low loss threshold, then the cost of the intrusion is low, while an Intruder Residence Time greater that the high loss threshold will lead to near max loss. The steep slope between the two thresholds indicates that it is necessary to limit the Intruder Residence Time to less than low loss threshold. The low loss threshold reflects the reality that it takes a certain time window for a hacker to be able to issue malicious commands, exploit backdoors, install Trojan horse programs, and/or steal/destroy data. A conservative estimate of the low loss threshold is in the range of minutes. Although there is no hard data for building the loss curve, there are reports that can help the process of building such a curve. For example, in [5] it is reported that in the context of on-line banking, security experts believe that a theft of $5,000 to $10,000 can be carried out over a few weeks, while larger losses up to $1 million are likely to take four to six months. In this context it is emphasized that the loss curve must account for the possibility that the *Intruder Residence Time* is spread over more than one successful breach of the system.

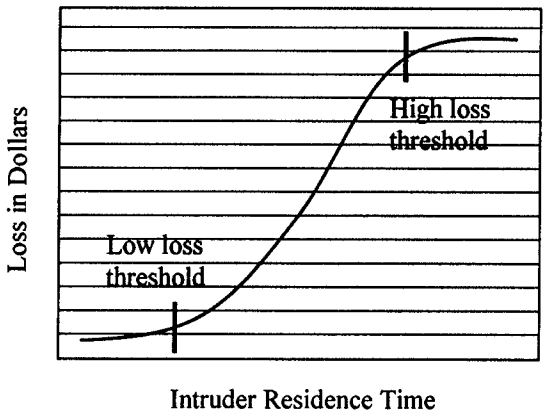


Figure 2: Loss curve: Loss in dollars vs. Intruder Residence Time in minutes

Lastly, we point out that SCIT complements and strengthens existing intrusion prevention and detection technologies [1,2]; we do not eliminate the use of the current intrusion management systems, but rather add another layer of defense, extending the idea of system "defense-in-depth" through periodic system cleansing. The effectiveness of SCIT depends on fast self-cleansing cycles, restricting the attackers to a very short time window to breach the system and cause harms.

The remainder of the paper is organized as follows. In Section 2, we present our experiences in building a prototype of SCIT firewalls. In Section 3, we discuss the feasibility of "SCIT-izing" more complicated systems, such as file servers. We give concluding remarks and outline future work in Section 4.

2. SCIT FIREWALLS

We chose firewalls as the first application of SCIT. The rational is twofold. First, the operation of stateless firewalls lends itself to SCIT, owing to the relative ease for a backup or mirror system to take over without disrupting ongoing traffic. Second, firewalls form the first line of defense for many private networks and thus are obvious targets of intrusion attacks. Strengthening the defense of firewalls significantly reduces the risk of security breaches in the whole network. The applications of the SCIT approach to more complex systems, such as NFS, DNS, and Web servers, will be discussed later.

Our testbed is based on the Virtual Machine software from VMWare, Inc. [6]. The virtual machine technology enables multiple *guest operating systems* to be installed and executed on top of a *host operating system*. In our demo, a client uses the Explorer on Windows 2000 to surf the web.

The client Windows is protected by SCIT firewalls. Both the client Windows machine and its SCIT firewalls are implemented as *virtual machines*. The underlying host machine is a Pentium 4 PC running RedHat 7.2 Linux. VMWare supports not only virtual machines but also *virtual networks*, which are emulated switched Ethernet networks. As seen in Figure 3, we use this feature to build two virtual networks, one with subnet ID 192.168.181.0/24 and the other 192.168.202.0/24. The first subnet connects the host system to the two firewalls. Both firewalls use a specialized version of RedHat 7.2 Linux. The latter subnet connects the firewalls to the client Windows. Inbound traffic are received by the host and relayed to one of the firewalls, which filters and relays the traffic to the client Windows. A third subnet, 192.168.200.0/24, is used by the two firewalls to probe each other. To enable the client Windows to communicate with the public Internet, its private IP address is translated to a public IP address by IP Masquerading, a form of network address translation that is supported by Linux kernels.

When a newly cleansed firewall is ready for operation, it must take over the IP addresses used by the presently running firewall. The firewall achieves this by issuing Gratuitous ARP messages using the Fake package [7]. The firewall rules are implemented in IPCHAINS. A shell script that executes the following steps controls the operations of each firewall. In the script, the first step is executed immediately after the underlying firewall has completed rebooting.

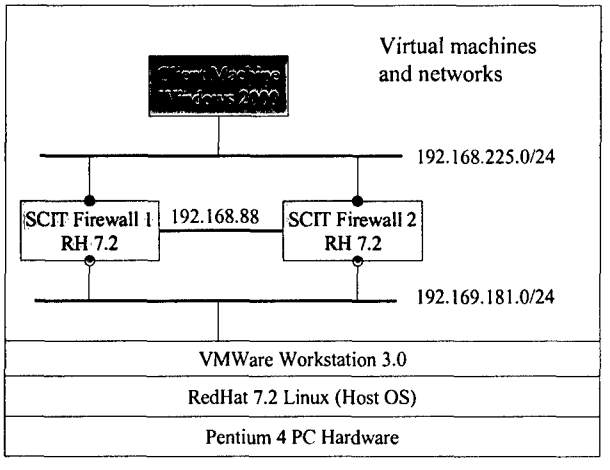


Figure 3. SCIT firewall prototype

1. Setup firewall rules using the `ipchains` command. Start traffic filtering and relaying in the background. The two tasks will continue to be performed in the background until the machine is shutdown for rebooting in Step 5.
2. Broadcast on subnets 192.168.225.0/24 and 192.168.181.0/24 Gratuitous ARP messages to announce the ownership of firewall IP addresses. This step lasts 10 seconds, one ARP message per second.
3. Wait for 10 seconds. This delay gives the other firewall extra times to detect the activities of this firewall and to reboot.
4. This firewall now assumes the other firewall is rebooting. It sends ping messages periodically over the 192.168.88.0 network to probe the other firewall, until a reply message is received. The receipt of a reply indicates the other firewall has completed self-cleansing and is ready to take over.
5. Reboot (and thus return to step 1 after completion).

From the client inside the SCIT firewall, the user opens a browser window and begins web surfing. We open two additional windows indicating which of the firewalls is operational. In the screen capture shown in Figure 4, these correspond to the black background windows on the top of the screen. The left-upper window shows that the presently running firewall is probing (unsuccessfully) the other firewall. That is, the firewall is executing the Step 4 given above. The right-upper window shows the booting message of the second firewall. The client Windows 2000 displays the CNN home page at the bottom. The page is, of course, obtained through the SCIT firewalls.

In general, for a typical HTML encoded web page, a firewall switch is barely perceptible. Examination of the trace sometimes shows occasional losses of packets when switching firewalls, but it appears that the retransmission of the packets is fast enough that the user cannot perceive the difference. Indeed, the setup is good enough for our own production uses – our research assistant regularly uses the setup for emails and web serving. With the setup, we observed self-cleansing cycles in the vicinity of 90 seconds.

Finally, it is worth pointing out that the self-cleansing of a firewall in the above steps comprises merely rebooting it. Assuming that the firewall is booted entirely from read-only storage, rebooting is sufficient to bring it to a clean state. While this assumption is reasonable for relatively simple devices like firewalls, in general cases more involved self-cleansing procedures are needed. For instance, using a tool called Tripwire, a system audit can be carried out after rebooting to check the integrity of system files [8].

3. A DISCUSSION OF SCIT SERVERS

We have shown a prototype design of SCIT firewalls. In this section we discuss the possibilities and difficulties of extending the concept to various type of servers in distributed computing environments. We call this task the "*SCIT-ization*" of servers.

1. Stateless servers are relatively straightforward to SCIT-ize. By stateless we mean the server does not

have to keep track of in memory the outcomes of previous tasks in order to carry out new tasks. NFS is a prominent example of stateless systems. Notice that dependences on the previous outcomes maintained in nonvolatile storage can be managed by SCIT, for mirror systems can share the storage. Storage sharing can be achieved by, for example, a SCSI bus in a small-scale system or a system-wide network (SAN) in a large cluster.

2. Servers that handle short sessions are relatively straightforward to SCIT-ize. Such servers are typically transaction oriented and process request-and-response types of tasks. Examples include DNS servers, some database servers, and certification servers. Telnet, FTP, and many application proxy servers are examples of long-session servers. A long session in a SCIT system needs to be migrated to a mirror system in the middle of the session. The task involves at a minimum moving endpoints of TCP connections on the fly and is unfeasible with the standard TCP. We will not further consider long-session servers for the time being.
3. Servers that manage static or semi-static data are relatively easy to SCIT-ize. A DNS, LDPA, or certification server, for instance, handles datasets that are typically small and infrequently changed. Static, small datasets enable efficient data mirroring and thus facilitates the construction of identical servers to operate alternatively. Due to the critical roles played by DNS and certification servers, SCIT technologies specifically developed for these servers further strengthen overall system security.

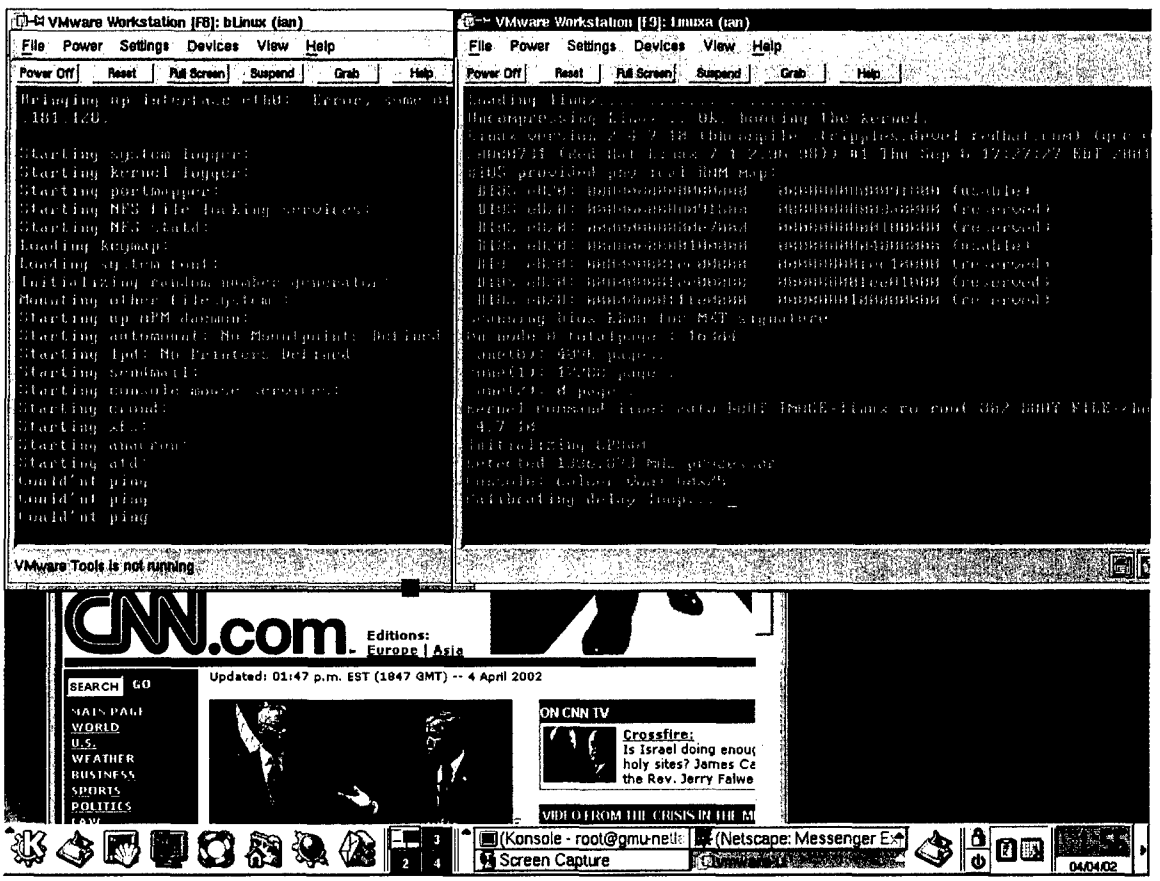


Figure 4. Screen capture of the SCIT firewall prototype

Indeed, we do not expect that the concept of SCIT be compatible with all types of computing systems. We do believe that SCIT is applicable to many important servers in distributed computing environments or Internet services, such as file systems, web servers, DNS servers, and certification services. The most important challenge in our future research is to design efficient SCIT architectures for these servers. To conclude this section, we present the

blueprint of our next SCIT system, a SCIT NFS server. The design is based on a high-availability file server architecture discussed in [9,10]. As shown in Figure 5, two server machines connect to a SCSI bus to share a SCSI hard drive that stores the file system data. Similar to the SCIT firewalls, the two server boxes must share one IP address using the technique described earlier. Also, a second set of network interfaces is used by currently running server to probe the status of the other server. At the time of this writing, we are implementing the design using the virtual machine technology.

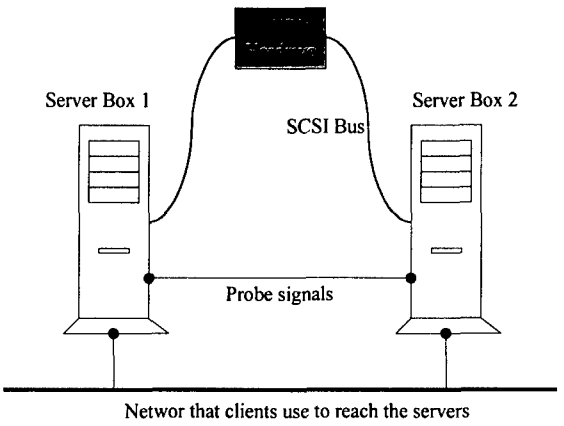


Figure 5. The blueprint of a SCIT NFS Server.

4. CONCLUSION

We have presented a novel application of high-availability computing, namely, intrusion containment. Our SCIT approach uses multiple, identical servers to execute in turn, allowing off-line servers to be checked for integrity and cleansed to return to a clean state. These self-cleansing activities occur periodically, regardless the presence/absence of intrusion alarms. As such, SCIT provides a defense against unknown or severe attacks that defeat the intrusion detection system. The effectiveness of SCIT depends on fast self-cleansing cycles, restricting the attackers to a very short time window to breach the system and inflict damages. The cost of hardware redundancy in SCIT systems can be avoided by using the virtual machine technology, as demonstrated in our SCIT firewall prototype.

At this early stage, we investigated only one self-cleansing method in detail, that is, rebooting followed possibly by data integrity checks and system audits. However, we envision many layers of cleansing activities in an ultimate SCIT system. In addition to rebooting the servers, one can kill and re-launch the server daemon. This process-level cleansing imposes less overhead, compared to system rebooting. Yet another system cleansing method is to re-load dynamic kernel modules, in the attempt to clean up those kernel codes potentially contaminated by hostile communications. With self-cleansing activities occurring at several levels of the system and at different frequencies, SCIT makes it very difficult for attackers to cause actual harms, even if they are able to penetrate existing intrusion defenses.

References

[1] M. Bishop, "Vulnerabilities Analysis," Recent Advances in Intrusion Detection, September 1999.

[2] Common Intrusion Detection Framework, http://www.gidos.org/.

[3] High-Availability Linux Project. Home page http://linux-ha.org/.

[4] Peter S. Weygant, *Clusters for High Availability*, Prentice Hall, 1996.

[5] Sandeep Junnarkar, "Anatomy of a hacking", available at http://news.com.com/2009-1017-893228.html, May 2002.

[6] VMware Inc. Home page http://www.vmware.com/.

[7] The Fake package. Home page http://vergenet.net/linux/fake/.

[8] Gene H. Kim and Eugene H. Spafford, "Writing, Supporting, and Evaluating Tripwire: A Publicly Available Security Tool," in *Proceedings of USENIX Applications Development Symposium*, (Toronto, Canada), April 1994. Also see http://www.tripwire.com/.

[9] Steve Blackmon and John Nguyen, "High-Availability File Server with Heartbeat," *System Admin, the Journal for UNIX Systems Administrators*, vol. 10, no. 9, September 2001.

[10] Richard Rabbat, Tom McNeal and Tim Burke, "A High-Availability Clustering Architecture with Data Integrity Guarantees," Proceedings of IEEE International conference on Cluster Computing, pages 178 – 182, (Newport Beach, California) October, 2001.